A parallel, object-oriented framework for frequency-domain wavefield imaging and inversion.

Tristan van Leeuwen and Felix J. Herrmann, Dept. of Earth and Ocean Sciences,University of British Columbia*

SUMMARY

We present a parallel object-oriented matrix-free framework for frequency-domain seismic modeling, imaging and inversion. The key aspects of the framework are its modularity and level of abstraction, which allows us to write code that reflects the underlying mathematical structure and develop unit-tests that guarantee the fidelity of the code. By overloading standard linear-algebra operations, such as matrix-vector multiplications, we can use standard optimization packages to work with our code without any modification. This leads to a scalable testbed on which new methods can be rapidly prototyped and tested on medium-sized 2D problems.
Although our current implementation uses (parallel) Matlab, all of these design principles can also be met by using lower-level languages which is important when we want to scale to realistic 3D problems.
We present some numerical examples on synthetic data.

INTRODUCTION

Seismic imaging and inversion relies heavily on complicated algorithms and complex mathematical concepts from different fields of expertise. A successful seismic inversion framework needs to combine a modeling engine (e.g., finite differences or finite elements), (non-) linear optimization algorithms and facilitate i/o of large amounts of data and the corresponding meta data. Due to the large scale of typical problems, a computer code that implements these algorithms and concepts has to be extremely efficient, scalable and robust. On the other hand, the interdisciplinary nature of these projects requires that experts from different fields can contribute to the code and that it can be easily verified that the code is correct. The challenge is in meeting these, seemingly mutually exclusive, requirements.

A necessary (but not sufficient) condition is that the basic building blocks of the code, such as the modeling operator and its Jacobian and their basic operations (forward modeling, action of the Jacobian and its adjoint) are exposed. This allows the units to be tested separately and re-used for different purposes. Then, these building blocks should be cast in pre-defined classes, such as (non-) linear operators and functionals. This allows us to have our code reflect the underlying mathematical principles. We do not have to sacrifice efficiency when following these principles because the individual blocks can still contain highly optimized code.

An added bonus of such a framework is that is allows for rapid prototyping and increases reproducability and accountability. Because the separate building blocks expose the underlying mathematics, it should be relatively straightforward to understand and implement the algorithm using a different modeling kernel (given that is also exposes the basic functionality as described above). Unit tests can ensure that any given implementation is correct.

For further reading on this design philosophy we refer to Claerbout and Fomel (2012) and Symes et al. (2011) who illustrates the approach for time- stepping codes and C++. A few existing libraries that facilitate such approaches to various degrees are PETSC (Balay et al., 1997), Trilinos (Heroux et al., 2005), the Rice Vector Library (Padula et al., 2009) and SPOT (van den Berg and Friedlander, 2009).

To illustrate our vision on how we can meet the above sketched criteria, we present an *object-oriented*, *matrix-free* framework for frequency- domain seismic modeling, imaging and inversion in parallel Matlab. The Matlab code is available for academic purposes and sponsors of the SLIM consortium. See `www.slim.eos.ubc.ca->software` for a preview.

The outline of the paper is as follows. First, we discuss the modeling kernel, which consists of the forward modeling operator and its Jacobian. Then we show how we use a framework for matrix-free linear algebra to implement functionality needed for various imaging and inversion purposes. We discuss how the different units of code should be tested and finally, we give some numerical examples on 2D synthetic data.

MODELING

We model the data by solving the Helmholtz equation and sampling the data at the receiver locations. We use a 9-point mixed-grid stencil (Jo, 1996) with a damping boundary layer to discretize the Helmholtz equation and denote the discretized system as

$$H(\mathbf{m})\mathbf{u} = {P_s}^T\mathbf{q}, \quad (1)$$

$$\mathbf{d} = P_r\mathbf{u}, \quad (2)$$

where P_s interpolates from the computational grid to the source grid —and hence the adjoint injects the source $\mathbf{q}$ into the computational grid— and P_r interpolates from the computational grid to the receiver locations. Injecting the sources in this manner ensures that the discretized point source behaves like a delta function (Anders Petersson and Sjögreen, 2010).

We define the modeling operator as $F(\mathbf{m},\mathbf{q}) = P_r H(\mathbf{m})^{-1}{P_s}^T\mathbf{q}$. Its Jacobian is given by

$$J = -P_r H^{-1} G, \quad (3)$$

where the matrix G is given by

$$G(\mathbf{m},\mathbf{u}) = \frac{\partial H(\mathbf{m})\mathbf{u}}{\partial \mathbf{m}},$$

and the wavefield $\mathbf{u}$ is given by solving 1. Of course, we never explicitly construct the Jacobian as a matrix, but instead calculate its action on a vector $\mathbf{x}$ by solving $H\mathbf{v} = G\mathbf{x}$, which gives us $J\mathbf{x} = P_r\mathbf{v}$.

The adjoint of the Jacobian follows immediately from the definition in eq. 3. Note that this requires an implementation of the adjoint of the interpolation operator.

IMPLEMENTATION

In order to reflect the underlying linear algebra in our implementation, we use SPOT, an object-oriented framework for matrix-free linear algebra in Matlab (van den Berg and Friedlander, 2009) . This allows us to implement matrix-free linear operators that behave like matrices in Matlab. Such an approach can also be realized in lower-level programming languages (Balay et al., 1997; Heroux et al., 2005; Padula et al., 2009; Symes et al., 2011).

Inverting the 2D Helmholtz matrix is done by direct factorization, for which Matlab uses optimized (external) libraries. Operations like interpolation can be done very efficiently in Matlab by writing them in terms of sparse matrix-vector products in order to avoid loops. Any remaining bottle-necks can be resolved by implementing certain algorithms in a lower-level language and wrapping them in MEX files.

The code is parallelized over frequencies. The Parallel Computing Toolbox provides distributed data types and basic operations such as matrix-vector multiplication and parallel for-loops. A parallel extension of SPOT, which was developed in our group (Modzelewski et al., 2010), is build on top of this and allows us to run all the code in parallel *without any modification to the solvers and optimization code*.

The modeling operator is implemented as a function `F(m,Q,model)` where `m` is the gridded model as vector, `Q` is a matrix where each columns represents a gridded source function and `model` contains various (acquisition) parameters. For example, a least-squares migration for background model `m0` and data `d` could be done as follows

```
% construct SPOT operator
J   = opDF(m0,Q,model);
% reconstruction with lsqr
dm  = lsqr(J,d);
```

Remember that `J` is *not* an explicit matrix, and construction of the operator does not require any computation. All the computations are done when multiplying with a vector. This allows us to use *any* solver in Matlab that aims to solve $A\mathbf{x} = b$ (possibly with regularization) and requires only matrix-vector products.

The least-squares misfit can be implemented as a matlab function:

```
function [f,g] = misfit(m,Q,Dobs,model)

[Dpred, J] = F(m,Q,model);
f          = .5*norm(Dpred - Dobs)^2;
g          = J'*(Dpred - Dobs);
```

where `f` is the function value and `g` is the gradient. The convention that non-linear functions return their function value and gradient (or vector and Jacobian) allows us to adopt ideas from algorithmic differentiation (AD) to calculate gradients of nested functions. See Neidinger (2010) for a nice introduction to AD and object-oriented programming in Matlab. For example, the misfit for a general penalty function `penalty` and model parametrization $\mathbf{m} = \mathbf{m}_0 + B\mathbf{x}$, where $\mathbf{m}_0$ encodes a reference model and B is particular basis (e.g., a subset of the computational grid or splines) looks like

```
function [f,g] = misfit(m0,x,...

[Dpred, J] = F(m0 + B*x,Q,model);
[f,df]     = penalty(Dpred - Dobs);
g          = B'*J'*df;
```

Many existing black-box optimization codes implemented in Matlab require functions of this form and can be used immediately without modification.

It is clear that this approach may lead to inefficiencies. For example, the combined action of the Jacobian and its adjoint requires 4 PDE solves if we use `J'*J`. A more efficient, direct, implementation would require only 3 PDE solves. We can implement this more efficient operator and test its validity against `J'*J`. The same holds for the least-squares misfit. The implementation described above requires 3 PDE solves, whereas a more efficient implementation would require only 2. Again, we can implement the more efficient version of the code and test it against the version of which we are sure that it is correct.

TESTING AND BENCHMARKING

The accuracy of the modeling operator can be tested against analytic solutions. Figure 1 (a) shows the error between the analytic and numerical solution for a constant-velocity of 2000 m/s and a frequency of 10 Hz as a function of the grid-spacing. Figure 1 (b) shows the analytic and numerical solution for a linear velocity profile $v = 1500 + 0.7z$ and a frequency of 10 Hz (cf. Kuvshinov and Mulder, 2006).

The modular implementation described above allows us to easily test the components. An important test for linear operators is the *dot-test*, which tests whether the implementation of the adjoint obeys the formal definition of the adjoint:

$$\langle A\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, A^* \mathbf{y} \rangle. \tag{4}$$

One typically looks at the ratio of the two for random vectors $\mathbf{x}, \mathbf{y}$ that are not in the null-space of A and A^* respectively. This ratio should be one up to machine precision (or very close). The result of such a test on our Jacobian is shown in Table 2. For this test we used the complex output of the adjoint of the Jacobian. For all the other examples we use only the real part.

For non-linear functions the *gradient-test* is an important unit-test. It is based on the Taylor expansion:

$$f(\mathbf{x} + h\delta\mathbf{x}) = f(\mathbf{x}) + h\langle \nabla f(\mathbf{x}), \delta\mathbf{x} \rangle + \mathscr{O}(h^2). \tag{5}$$

We can test whether the error decays as predicted. A similar argument can be be used to devise a test for the Jacobian. In that case, care must be taken to select a vector $\delta\mathbf{x}$ that is not in the null-space of the Jacobian.

Results of these tests for our Jacobian and least-squares misfit are presented in figure 2

To illustrate the scalability of the code we compute data for a 344×1144 model for 64 frequencies and 500 sources using 2,4,8,16 and 32 processors. The computations where done on an IBMx3550 cluster where each node has 2 quad core 2.6 GHz Intel CPUs, 16 GB of memory, Voltaire Infiniband x4 inter-processor network and a 1Gb Ethernet connection. We use 2 CPUs per node. The timing, speedup and efficiency (speedup / no. of CPUs) of a single run are shown in Table 1.

# of procs	time [s]	speed up	efficiency
2	36611	1.58	0.79
4	19544	2.96	0.74
8	9623	6.03	0.75
16	4525	12.82	0.80
32	2289	25.34	0.79

Table 1: Timing, speed up and efficiency of modeling operator for 500 sources, 64 frequencies for a 1140× 344 model on different number of CPU's.

$\langle A\mathbf{x},\mathbf{y}\rangle$	$\langle \mathbf{x},A^*\mathbf{y}\rangle$
-9.3405e+08+4.1677e+08i	-9.3405e+08+4.1677e+08i
8.8128e+08-1.6759e+08i	8.8128e+08-1.6759e+08i
3.7126e+08+2.1559e+07i	3.7126e+08+2.1559e+07i
-1.7585e+08-1.7851e+08i	-1.7585e+08-1.7851e+08i

Table 2: Results of the adjoint-test of the Jacobian. The ratio between the left and right columns is $1 \pm \mathcal{O}(10^{-15})$. Note that for this test we used complex in and output of our Jacobian. In practice, the adjoint would only return the real part.

EXAMPLE I: LEAST-SQUARES MIGRATION

We perform a least-squares migration on a subset of the Marmousi model. We use 21 equispaced sources and 401 equispaced receivers located at the top of the model and frequencies 4-25 Hz with 0.5 Hz spacing. To solve the system $J\mathbf{x} = \mathbf{d}$ we use Matlab's `lsqr`, see also the section on Implementation. The result after 50 iterations is depicted in figure. Finally, we repeat the experiment with a Jacobian that does not pass the dot-test (the ratio is $1 \pm \mathcal{O}(10^{-2})$); instead of the adjoint interpolation ${P_r}^T$, we use a forward interpolation to interpolate from the data to the grid. The result is depicted in figure 3. Even such a seemingly small error in the adjoint leads to a severe degradation of the result.

EXAMPLE II: FWI

We reproduce the famous Camembert example (Gauthier, 1986). The true model is depicted in figure 4 (a). We use a reflection setup with 11 equi-spaced sources and 101 equispaced receivers at the top of the model and a transmission setup with 11 equispaced sources on the left and 101 equispaced receivers on the right of the model. For both we use frequencies 5, 10 and 15 Hz. simultaneously. For the optimization we use an L-BFGS method with a weak Wolfe linesearch (cf. Nocedal and Wright, 2000). We start the optimization from a constant velocity of 2500 m/s and run the optimization for 100 iterations. The results are depicted in figures 4 (b) and (c). Finally, we repeat the experiment with a *wrong* Jacobian: we use a discretization of the linearized system – rather than a linearization of the discretized system – leading to $G(\mathbf{m},\mathbf{u}) = \omega^2\mathsf{diag}(\mathbf{u})$ and instead of the adjoint interpolation ${P_r}^T$, we use a forward interpolation to interpolate from the data to the grid. The resulting least-squares function does not pass the gradient-test. This causes the optimization method to stall after about 20 iterations because it cannot decrease the objective any further. This is illustrated in figure 4 (d).

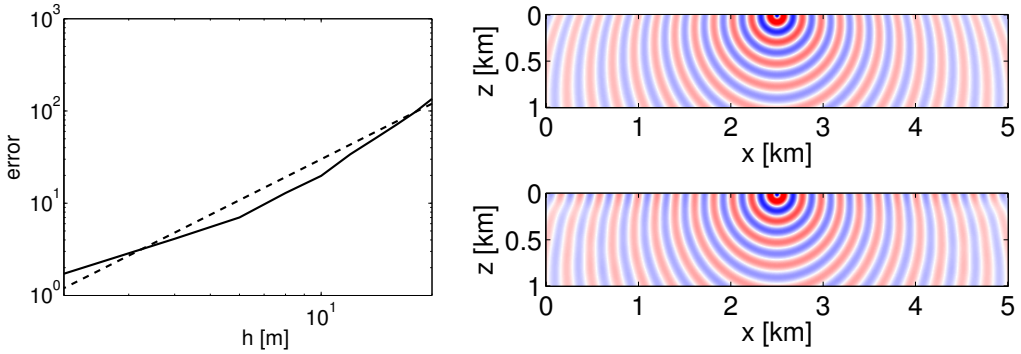
CONCLUSION

We have presented a parallel, object-oriented framework for frequency-domain wavefield modeling, imaging and inversion. The code is implemented in Matlab and will be made available for acamedic purposes. The main take-out messages of this paper are the following:

- By overloading basic operations such as matrix-vector multiplies and separating the parameters from the linear algebra (e.g., using coordinate-free operators) we can utilize standard optimization codes. It allows us to expose the underlying mathematical structure of the code, which facilitates reproducability and dissemination of computational research.
- Efficiency bottlenecks in the code can be resolved by optimizing particular modules or combinations of them and the unit-tests can be used to assure that the code is still correct.
- Such an approach is by no means tied to high-level languages such as Matlab and can be realized using any programming language that supports object-oriented programming. There are many existing libraries that implement these ideas.
- Black-box optimization codes tend to work best when using accurate adjoints and gradients. Failure to pass the dot and/or gradient-test may lead to dramatic degradation of the results.

ACKNOWLEDGEMENTS

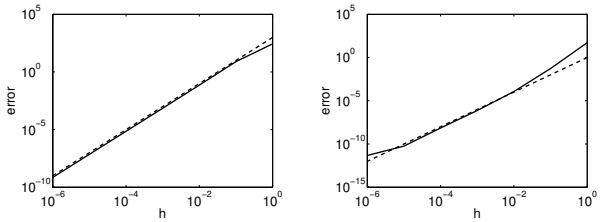
This work was in part financially supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant (22R81254) and the Collaborative Research and Development Grant DNOISE II (375142-08). This research was carried out as part of the SINBAD II project with support from the following organizations: BG Group, BGP, BP, Chevron, ConocoPhillips, Petrobras, PGS, Total SA, and WesternGeco.



(a) (b)

Figure 1: (a) Error between numerical and exact solution for a constant velocity as a function of the gridspacing. The dashed line shows quadratic decay. (b) Exact (top) and numerical (bottom) solution for linear velocity profile $1500+0.7z$ m/s at 10 Hz.

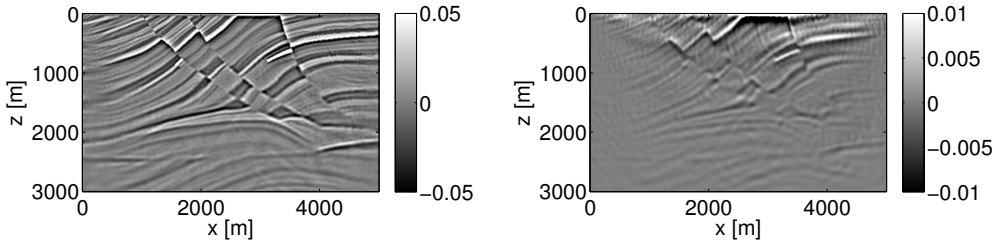
.



(a) (b)

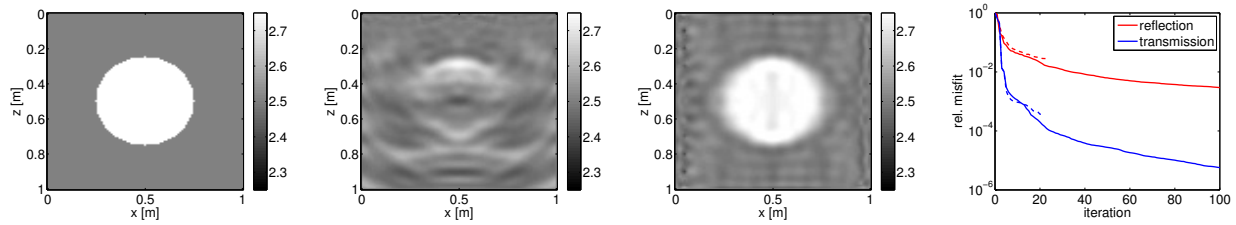
Figure 2: Error for (a) Jacobian and (b) least-squares misfit. The dashed line shows the desired decay of $\mathcal{O}(h^2)$

.



(a) (b)

Figure 3: least-squares migration of part of the Marmousi model. Shown are the results after 50 LSQR iterations using the exact adjoint (a) and an approximate adjoint (b). Note the difference in colorscale.



(a) (b) (c) (d)

Figure 4: FWI results on the Camembert model, depicted in (a). The results after 100 L-BFGS iterations for reflection and transmission data are shown in (b) and (c). Convergence plots are show in (d). The dashed lines show the result when an approximate Jacobian was used to compute the gradient.

REFERENCES

Anders Petersson, N., and B. Sjögreen, 2010, Stable grid refinement and singular source discretization for seismic wave simulations.: communications in computational physics, **8**, 1074–1110.

Balay, S., W. D. Gropp, L. C. McInnes, and B. F. Smith, 1997, Efficient management of parallelism in object oriented numerical software libraries: Modern Software Tools in Scientific Computing, Birkhäuser Press, 163–202.

Claerbout, J., and S. Fomel, 2012, Image estimation by example: Geophysical soundings image construction.

Gauthier, O., 1986, Two-dimensional nonlinear inversion of seismic waveforms: Numerical results: Geophysics, **51**, 1387.

Heroux, M. A., R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, 2005, An overview of the trilinos project: ACM Trans. Math. Softw., **31**, 397–423.

Jo, C.-H., 1996, An optimal 9-point, finite-difference, frequency-space, 2-D scalar wave extrapolator: Geophysics, **61**, 529.

Kuvshinov, B. N., and W. a. Mulder, 2006, The exact solution of the time-harmonic wave equation for a linear velocity profile: Geophysical Journal International, **167**, 659–662.

Modzelewski, H., T. Lai, and S. Jalali, 2010, pSPOT, a parallel extension of SPOT. (https://github.com/slimgroup/pSPOT).

Neidinger, R. D., 2010, Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming: SIAM Review, **52**, 545.

Nocedal, J., and S. Wright, 2000, Numerical optimization: Springer.

Padula, A. D., S. D. Scott, and W. W. Symes, 2009, A software framework for abstract expression of coordinate-free linear algebra and optimization algorithms: ACM Transactions on Mathematical Software, **36**, 1–36.

Symes, W. W., D. Sun, and M. Enriquez, 2011, From modelling to inversion: designing a welladapted simulator: Geophysical Prospecting, **59**, 814–833.

van den Berg, E., and M. P. Friedlander, 2009, Spot A Linear-Operator Toolbox. (http://www.cs.ubc.ca/labs/scl/spot/).